



**【実習】 μ T-Kernel 入門
(協力：ルネサス エレクトロニクス)**

プログラミング演習

目次

第1章 エミュレータの接続とプログラムの実行

1.1	演習環境とボード構成	1-2
1.1.1	演習環境	1-2
1.1.2	演習フォルダ構成	1-2
1.1.3	ターゲットボードの構成	1-2
1.1.4	入出力装置のポート割り付け（演習で使用する I/O のみ）	1-3
1.2	E1 エミュレータとの接続とプログラム実行	1-5
1.3	μ T-Kernel のリソース情報表示	1-6

第2章 演習問題

演習1	タスク管理機能の演習	2-2
演習2	タスク付属同期機能の演習	2-5
演習3	セマフォの演習	2-8
演習4	イベントフラグの演習	2-11
演習5	メールボックスの演習	2-15
演習6	ミューテックスの演習	2-19
演習7	メッセージバッファの演習	2-22
演習8	固定長メモリプールの演習	2-26

第 1 章

エミュレータの接続とプログラムの実行

1.1	演習環境とボード構成	1-2
1.2	E1 エミュレータとの接続とプログラム実行	1-5

1.1 演習環境とボード構成

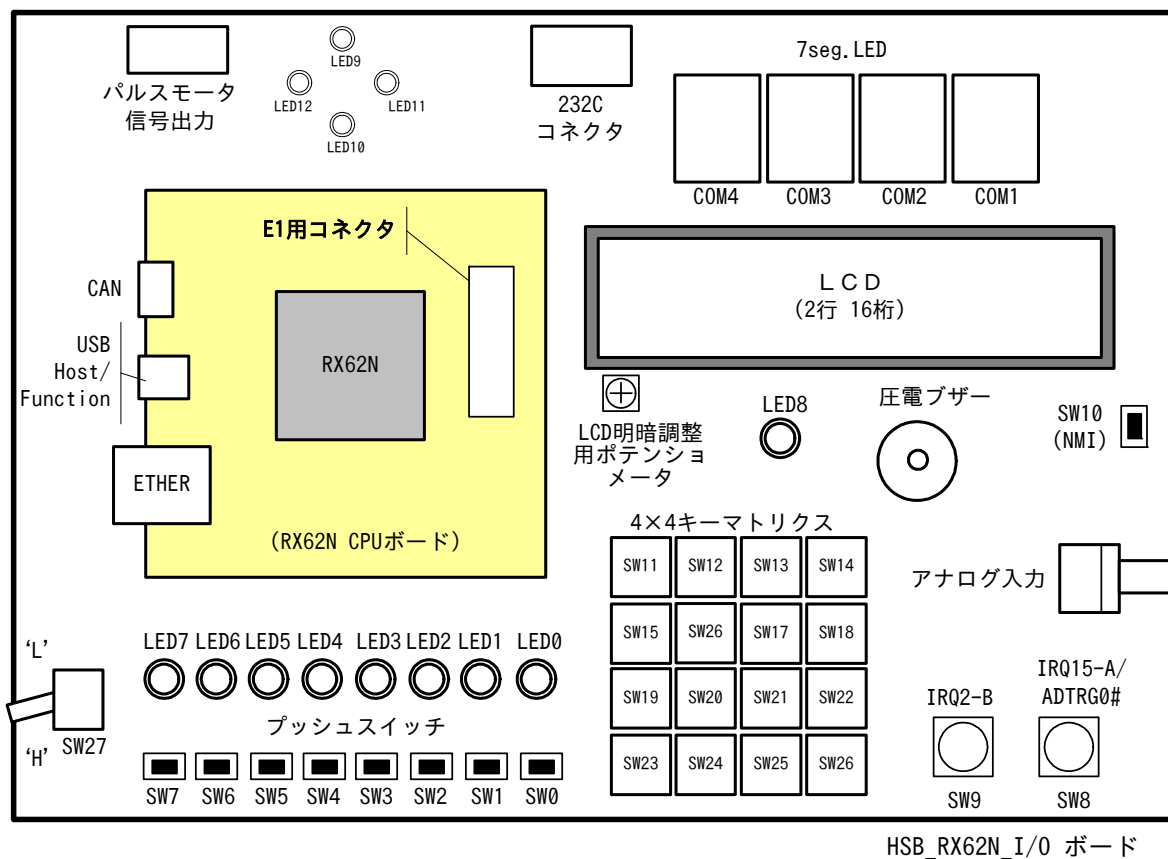
1.1.1 演習環境

- (1) ターゲットボード : HSBRX62N(CPU ボード) + HSB_RX62N_I/O(I/O ボード)
- (2) コンパイラ : RX C/C++ コンパイラ
- (3) エミュレータ : E1 エミュレータ

1.1.2 演習フォルダ構成

```
C:\workspace
|-utkernel_source_rx62n
|  |-ide
|    |-cs
|      |-01_task_manage.c
|      |-02_task_sync.c
|      |-03_semaphore.c
|      |-04_eventflag.c
|      |-05_mailbox.c
|      |-06_mutex.c
|      |-07_messagebuffer.c
|      |-08_memorypool.c
|      |-99_apl_sample.c
```

1.1.3 ターゲットボードの構成



1.1.4 入出力装置のポート割り付け（演習で使用する I/O のみ）

< LED0 ~ 7 >

接続端子	シンボル	アクティブ
PA_0	LED0	“L”
PA_1	LED1	“L”
PA_2	LED2	“L”
PA_3	LED3	“L”
PA_4	LED4	“L”
PA_5	LED5	“L”
PA_6	LED6	“L”
PA_7	LED7	“L”

< 5φLED >

接続端子	シンボル	アクティブ
P1_3/P1_5	LED8	“L”

< 7seg.LED >

接続端子	シンボル	アクティブ
PE_0	セグメント a	“H”
PE_1	セグメント b	“H”
PE_2	セグメント c	“H”
PE_3	セグメント d	“H”
PE_4	セグメント e	“H”
PE_5	セグメント f	“H”
PE_6	セグメント g	“H”
PE_7	セグメント Dp	“H”

接続端子	シンボル	アクティブ
P4_4	COM1(最右桁)	“H”
P4_5	COM2	“H”
P4_6	COM3	“H”
P4_7	COM4(最左桁)	“H”

<プッシュスイッチ>

接続端子	シンボル	入力レベル
PD_0	SW0	OFF= “H” , ON= “L”
PD_1	SW1	OFF= “H” , ON= “L”
PD_2	SW2	OFF= “H” , ON= “L”
PD_3	SW3	OFF= “H” , ON= “L”
PD_4	SW4	OFF= “H” , ON= “L”
PD_5	SW5	OFF= “H” , ON= “L”
PD_6	SW6	OFF= “H” , ON= “L”
PD_7	SW7	OFF= “H” , ON= “L”

<トグルスイッチ>

接続端子	シンボル	入力レベル
P1_7	SW27	上側に倒す→ “L” 、下側に倒す→ “H”

< IRQ 入力用スイッチ>

接続端子	シンボル	入力レベル
ADTRG0#/IRQ15-A/P0_7	SW8	OFF= “H” , ON= “L”
IRQ2-B/P1_2	SW9	OFF= “H” , ON= “L”

<アナログ入力>

接続端子	シンボル	入力レベル
AN0/P4_0	R22	時計回りで0V→5V入力

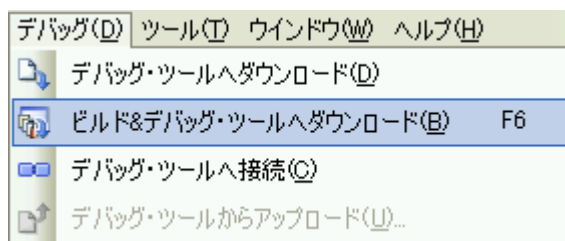
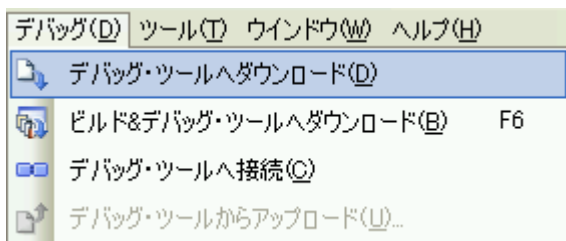
<シリアル>

接続端子	シンボル
TxD6/P0_0	—
RxD6/P0_1	—
SCK6/P0_2	—

1.2 E1 エミュレータとの接続とプログラム実行

(1) E1 エミュレータやシミュレータへのダウンロード

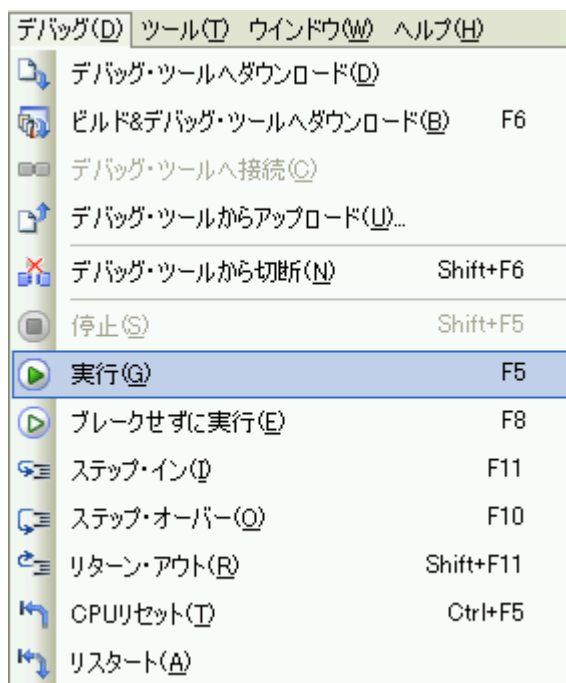
「デバッグメニュー」→「デバッグ・ツールヘダウンロード」、あるいは「ビルド&デバッグ・ツールヘダウンロード」、または「F6」でE1 エミュレータへのダウンロードが行えます。



(2) プログラムの実行と停止

「デバッグメニュー」→「実行」または「F5」でプログラムの実行が行えます。

「デバッグメニュー」→「停止」または「Shift + F6」でプログラムの停止が行えます。



1.3 μ T-Kernel のリソース情報表示

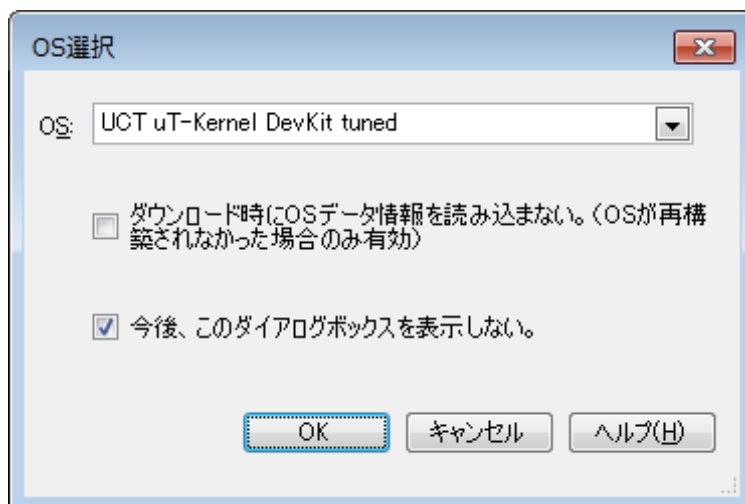
(1) リソース情報

CubeSuite+ には μ T-Kernel のリソース情報を表示する機能があります。

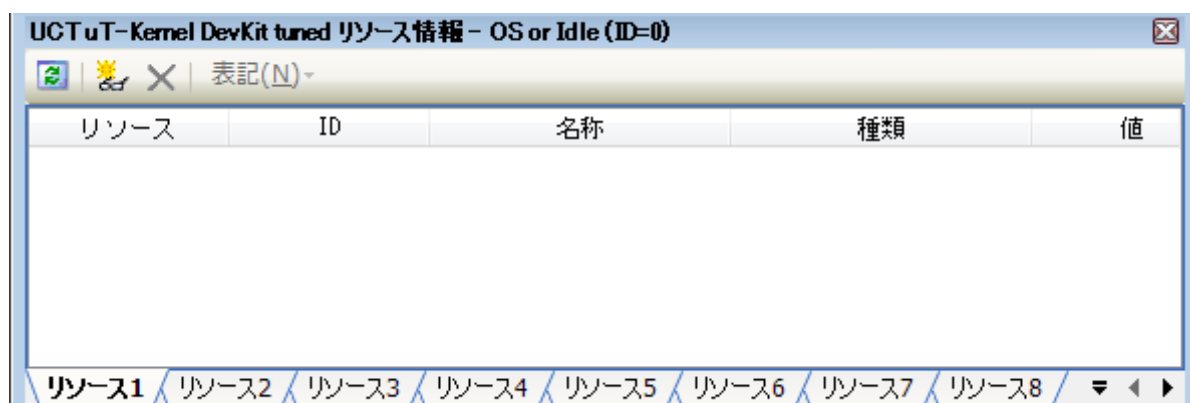
「表示メニュー」→「リアルタイム OS」→「リソース情報」を選択します。



表示された OS 選択のダイアログで「UCT uT-Kernel DevKit tuned」を選択してください。

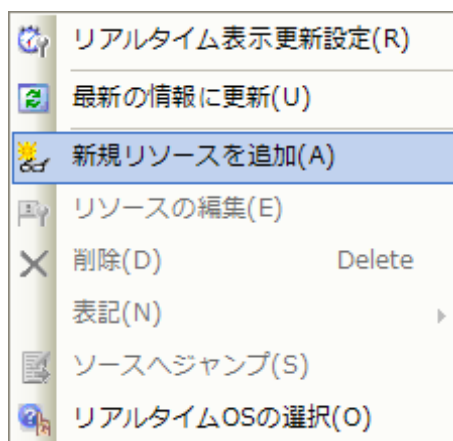


そうすると UCT uT-Kernel DevKit tuned のリソース情報のウィンドウが開かれます。

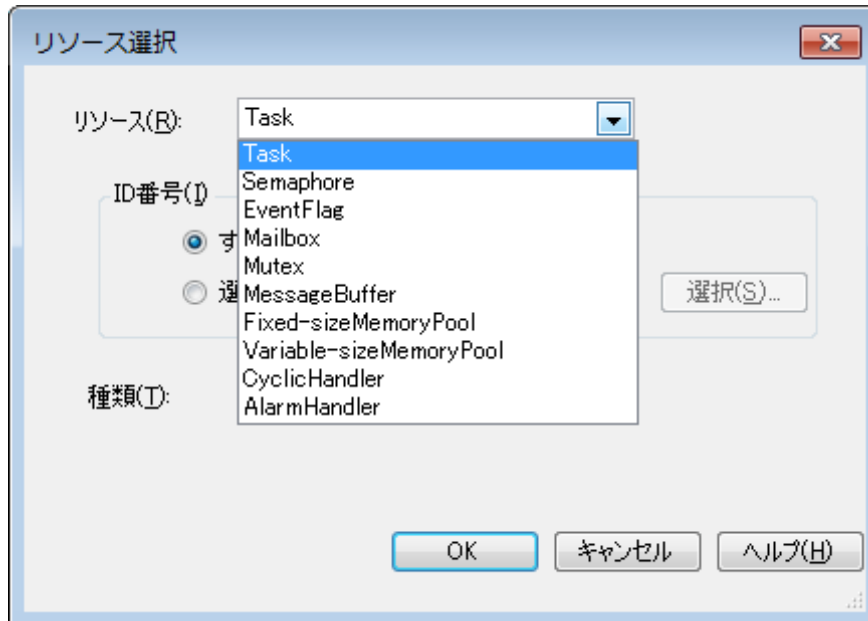


(2) リソースの追加

表示された UCT uT-Kernel DevKit tuned のリソース情報のウィンドウにリソースを追加する際は目的のウィンドウで右クリックし、「新規リソースの追加」を実行します。



表示されたリソース選択のダイアログ上で参照したいリソースを選択し、OK します。なお、選択したリソースはすべての ID を表示することと選択した ID だけを表示することが可能です。



以下はタスクのリソースを表示したものです。もし、すべての ID を選択した場合は μ T-Kernel のコンフィグレーションで指定されたすべての ID が表示されます。また、各リソースを生成した際、TA_DSNAME の属性を指定し、パラメータパケットの dsname に名称を設定したものは、指定した名称が表示されます。

UCTuT-Kernel DevKit tuned リソース情報 - (ID=1)

リソース	ID	名称	種類	値
Task	1		Status	RUNNING
Task	2		Status	NON-EXISTENT
Task	3		Status	NON-EXISTENT
Task	4		Status	NON-EXISTENT

リソース1 リソース2 リソース3 リソース4 リソース5 リソース6 リソース7 リソース8

第 2 章

演習問題

演習 1	タスク管理機能の演習.....	2-2
演習 2	タスク付属同期機能の演習.....	2-5
演習 3	セマフォの演習.....	2-8
演習 4	イベントフラグの演習.....	2-11
演習 5	メールボックスの演習.....	2-15
演習 6	ミューティクスの演習.....	2-19
演習 7	メッセージバッファの演習.....	2-21
演習 8	固定長メモリプールの演習.....	2-25

演習1 タスク管理機能の演習

- ・ tsk_a、tsk_b の2つのタスクから構成
- ・ tsk_a は SW8 (IRQ15) で起動
- ・ tsk_a から DORMANT 状態の tsk_b に対して、tk_sta_tsk を複数回発行した時の動作を確認する
- ・ tk_sta_tsk による起動要求にはキューイング機能がないことを確認する
- ・ tsk_a、tsk_b の優先度を変更することで動作タイミングが変わることを確認する

01_task_manage.c

```
#include <tk/tkernel.h>
#include <tm/tmonitor.h>
#include <libstr.h>
#include <machine.h>
#include "iodefine.h"

EXPORT void tsk_a(INT stacd, VP exinf);
EXPORT void tsk_b(INT stacd, VP exinf);
EXPORT void irq15_hdr(UINT dintno);

typedef enum { TSK_A, TSK_B, OBJ_KIND_NUM } OBJ_KIND;
EXPORT ID ObjID[OBJ_KIND_NUM];          // ID テーブル

EXPORT INT usermain( void )
{
    T_CTSK t_ctsk;
    T_DINT t_dint;
    ID objid;
    t_ctsk.tskatr = TA_HLNG | TA_DSNAME;
    t_ctsk.stksz = 1024;
    t_ctsk.task = tsk_a;                  // tsk_a の起動アドレス
    t_ctsk.itstkpri = 1;                  // tsk_a の優先度
    strcpy( (char *)t_ctsk.dsname, "tsk_a" );    // tsk_a の名称
    if( (objid = tk_cre_tsk( &t_ctsk )) <= E_OK ) {    // tsk_a の生成
        tm_putstring(" *** Failed in the creation of tsk_a.¥n");
        goto ERROR;
    }
    ObjID[TSK_A] = objid;
    tm_putstring("*** tsk_a created.¥n");

    t_ctsk.task = tsk_b;                  // tsk_b の起動アドレス
    t_ctsk.itstkpri = 2;                  // tsk_b の優先度
    strcpy( (char *)t_ctsk.dsname, "tsk_b" );    // tsk_b の名称
    if( (objid = tk_cre_tsk( &t_ctsk )) <= E_OK ) {    // tsk_b の生成
        tm_putstring(" *** Failed in the creation of tsk_b.¥n");
        goto ERROR;
    }
    ObjID[TSK_B] = objid;
    tm_putstring("*** tsk_b created.¥n");
}
```

```

t_dint.intatr = TA_HLNG;
t_dint.inthdr = irq15_hdr;
if( tk_def_int( VECT( ICU, IRQ15 ), &t_dint ) != E_OK ) {
    tm_putstring( " *** Failed in the definition of irq15_hdr.¥n" );
    goto ERROR;
}
tm_putstring( " *** irq15_hdr defined.¥n" );

PORTE.DR.BIT.B0 = 1; // 最上位のセグメントラインを設定
PORTE.DDR.BYTE = 0xFF; // セグメントラインをアクティブに設定
PORT4.DDR.BIT.B4 = PORT4.DR.BIT.B4 = 1; // 最右桁のLEDをアクティブに設定

PORT0.ICR.BIT.B7 = 1; // P07(IRQ15)の入力バッファをON
ICU.IRQCR[15].BIT.IRQMD = 1; // IRQ15を立ち下がりエッジに設定
IR( ICU, IRQ15 ) = 0; // IRQ15の割り込み要求フラグをクリア
IPR( ICU, IRQ15 ) = 1; // IRQ15の割り込みレベルを1に設定
IEN( ICU, IRQ15 ) = 1; // IRQ15の割り込みを許可

while( 1 ) wait( );
ERROR:
return 0;
}

EXPORT void tsk_a(INT stacd, VP exinf)
{
    int i;
    for( i=0 ; i<2 ; i++ ) {
        tm_putstring( " *** tk_sta_tsk to tsk_b.¥n" );
        if( tk_sta_tsk( ObjID[TSK_B], 0 ) != E_OK ) // tsk_bの起動
            tm_putstring( " *** Failed in tk_sta_tsk to tsk_b.¥n" );
    }
    tk_ext_tsk( ); // tsk_aの終了
}

EXPORT void tsk_b(INT stacd, VP exinf)
{
    int i;
    tm_putstring( " *** tsk_b is Running.¥n" );
    while( PORTE.DR.BYTE != 0x40 ) { // LEDのローテート処理
        for( i=0 ; i<12000000 ; i++ ) ; // 500msの待ち時間
        PORTE.DR.BYTE <<= 1;
    }
    PORTE.DR.BYTE = 0x01;
    tk_ext_tsk( ); // tsk_bの終了
}

EXPORT void irq15_hdr(UINT dintno)
{
    tk_sta_tsk( ObjID[TSK_A], 0 ); // tsk_aの起動
}

```

タスク管理機能のスケジューリング

優先度：タスクA \geq タスクB

事象	カーネル	タスクA	タスクB
		RUNNING	DORMANT
1回目のtk_sta_tsk⇒			READY
2回目のtk_sta_tsk⇒		E_OK	
		E_OBJ	
タスクAのtk_ext_tsk⇒		DORMANT	RUNNING
タスクBのtk_ext_tsk⇒			DORMANT

優先度：タスクA < タスクB

事象	カーネル	タスクA	タスクB
		RUNNING	DORMANT
1回目のtk_sta_tsk⇒		READY	RUNNING
タスクBのtk_ext_tsk⇒		RUNNING	DORMANT
2回目のtk_sta_tsk⇒		READY	RUNNING
タスクBのtk_ext_tsk⇒		RUNNING	DORMANT
タスクAのtk_ext_tsk⇒		DORMANT	

演習2 タスク付属同期機能の演習

- ・ tsk_a、tsk_b の2つのタスクから構成
- ・ tsk_a は SW8 (IRQ15) で起動
- ・ tsk_a から WAITING 状態の tsk_b に対して、tk_wup_tsk を複数回発行した時の動作を確認する
- ・ tk_wup_tsk による起床要求にはキューイング機能があることを確認する
- ・ tsk_a、tsk_b の優先度を変更することで動作タイミングが変わることを確認する

02_task_sync.c

```
#include <tk/tkernel.h>
#include <tm/tmonitor.h>
#include <libstr.h>
#include <machine.h>
#include "iodefine.h"

EXPORT void tsk_a(INT stacd, VP exinf);
EXPORT void tsk_b(INT stacd, VP exinf);
EXPORT void irq15_hdr(UINT dintno);

typedef enum { TSK_A, TSK_B, OBJ_KIND_NUM } OBJ_KIND;
EXPORT ID ObjID[OBJ_KIND_NUM];          // ID テーブル

EXPORT INT usermain( void )
{
    T_CTSK t_ctsk;
    T_DINT t_dint;
    ID objid;
    t_ctsk.tskatr = TA_HLNG | TA_DSNAME;
    t_ctsk.stksz = 1024;
    t_ctsk.task = tsk_a;                  // tsk_a の起動アドレス
    t_ctsk.itskpri = 1;                   // tsk_a の優先度
    strcpy( (char *)t_ctsk.dsname, "tsk_a" );    // tsk_a の名称
    if( (objid = tk_cre_tsk( &t_ctsk )) <= E_OK ) { // tsk_a の生成
        tm_putstring( " *** Failed in the creation of tsk_a. %n" );
        goto ERROR;
    }
    ObjID[TSK_A] = objid;
    tm_putstring( " *** tsk_a created. %n" );

    t_ctsk.task = tsk_b;                  // tsk_b の起動アドレス
    t_ctsk.itskpri = 2;                   // tsk_b の優先度
    strcpy( (char *)t_ctsk.dsname, "tsk_b" );    // tsk_b の名称
    if( (objid = tk_cre_tsk( &t_ctsk )) <= E_OK ) { // tsk_b の生成
        tm_putstring( " *** Failed in the creation of tsk_b. %n" );
        goto ERROR;
    }
    ObjID[TSK_B] = objid;
    tm_putstring( " *** tsk_b created. %n" );
    if( tk_sta_tsk( ObjID[TSK_B], 0 ) != E_OK )    // tsk_b の起動
        tm_putstring( " *** Failed in start of tsk_b. %n" );
}
```



```

    t_dint.intatr = TA_HLNG;
    t_dint.inthdr = irq15_hdr;
    if( tk_def_int( VECT( ICU, IRQ15 ), &t_dint ) != E_OK ) {
        tm_putstring( " *** Failed in the definition of irq15_hdr.¥n" );
        goto ERROR;
    }
    tm_putstring( " *** irq15_hdr defined.¥n" );
    PORTE.DR.BIT.B0 = 1; // 最上位のセグメントラインを設定
    PORTE.DDR.BYTE = 0xFF; // セグメントラインをアクティブに設定
    PORT4.DDR.BIT.B4 = PORT4.DR.BIT.B4 = 1; // 最右桁の LED をアクティブに設定
    PORT0.ICR.BIT.B7 = 1; // P07(IRQ15) の入力バッファを ON
    ICU.IRQCR[15].BIT.IRQMD = 1; // IRQ15 を立ち下がりエッジに設定
    IR( ICU, IRQ15 ) = 0; // IRQ15 の割り込み要求フラグをクリア
    IPR( ICU, IRQ15 ) = 1; // IRQ15 の割り込みレベルを 1 に設定
    IEN( ICU, IRQ15 ) = 1; // IRQ15 の割り込みを許可

    while( 1 ) wait( );
ERROR:
    return 0;
}

EXPORT void tsk_a(INT stacd, VP exinf)
{
    int i;
    for( i=0 ; i<2 ; i++ ) {
        tm_putstring( " *** tk_wup_tsk to tsk_b.¥n" );
        if( tk_wup_tsk( ObjID[TSK_B] ) != E_OK ) // tsk_b の起床
            tm_putstring( " *** Failed in tk_wup_tsk to tsk_b.¥n" );
    }
    tk_ext_tsk( ); // tsk_a の終了
}

EXPORT void tsk_b(INT stacd, VP exinf)
{
    int i;
    tm_putstring( " *** tsk_b started.¥n" );
    while( 1 ) {
        tm_putstring( " *** tsk_b is Waiting.¥n" );
        tk_slp_tsk( TMO_FEVR ); // tsk_b の起床待ち
        tm_putstring( " *** tsk_b is Running.¥n" );
        while( PORTE.DR.BYTE != 0x40 ) { // LED のローテート処理
            for( i=0 ; i<12000000 ; i++ ) ; // 500ms の待ち時間
            PORTE.DR.BYTE <= 1;
        }
        PORTE.DR.BYTE = 0x01;
    }
}

EXPORT void irq15_hdr(UINT dintno)
{
    tk_sta_tsk( ObjID[TSK_A], 0 ); // tsk_a の起動
}

```

タスク付属同期機能のスケジューリング

優先度：タスクA \geq タスクB

事象	カーネル	タスクA	タスクB
	0:キューイング数(タスクB)	RUNNING	WAITING
1回目のtk_wup_tsk⇒	0		READY
2回目のtk_wup_tsk⇒	1	E_OK	
tk_ext_tsk⇒		E_OK	
		DORMANT	RUNNING
tk_slp_tsk⇒	0		
tk_slp_tsk⇒	0		
			WAITING

優先度：タスクA<タスクB

事象	カーネル	タスクA	タスクB
	0:キューイング数(タスクB)	RUNNING	WAITING
1回目のtk_wup_tsk⇒	0	READY	RUNNING
tk_slp_tsk⇒	0	RUNNING	WAITING
2回目のtk_wup_tsk⇒	0	READY	RUNNING
tk_slp_tsk⇒	0	RUNNING	WAITING
tk_ext_tsk⇒		DORMANT	

演習3 セマフォの演習

- ・ tsk_a、tsk_b の2つのタスクと両者を周期的に起動する rand_tsk の3つから構成
- ・ tsk_a は最左桁の7セグメントLEDの上段を点灯する
- ・ tsk_b は2桁目の7セグメントLEDの下段を点灯する
- ・ 排他制御を行わないと7セグメントLEDの表示が正しく行えないことを確認する
- ・ セマフォにより排他制御を行えば7セグメントLEDの表示を正しく行えることを確認する

03_semaphore.c

```
#include <tk/tkernel.h>
#include <tm/tmonitor.h>
#include <libstr.h>
#include <machine.h>
#include "iodefine.h"

EXPORT void tsk_a(INT stacd, VP exinf);
EXPORT void tsk_b(INT stacd, VP exinf);
EXPORT void rand_tsk(INT stacd, VP exinf);
EXPORT unsigned int rand(void);

typedef enum { SEMID, TSK_A, TSK_B, RAND_TSK, OBJ_KIND_NUM } OBJ_KIND;
EXPORT ID ObjID[OBJ_KIND_NUM];          // ID テーブル

EXPORT INT usermain( void )
{
    T_CSEM t_csem;
    T_CTSK t_ctsk;
    ID objid;

    t_csem.sematr = TA_TFIFO | TA_DSNAME;          // セマフォ属性の設定
    t_csem.isemcnt = 1;                          // セマフォの初期値の設定
    t_csem.maxsem = 1;                            // セマフォの最大値の設定
    strcpy( (char *)t_csem.dsname, "led_sem" );    // セマフォの名称
    if( (objid = tk_cre_sem( &t_csem )) <= E_OK ) { // セマフォの生成
        tm_putstring(" *** Failed in the creation of semaphore.¥n");
        goto ERROR;
    }
    ObjID[SEMID] = objid;
    tm_putstring(" *** semaphore created.¥n");
}
```

```

t_ctsk.tskatr = TA_HLNG | TA_DSNAME;
t_ctsk.stksz = 1024;
t_ctsk.task = tsk_a; // tsk_a の起動アドレス
t_ctsk.itskpri = 2; // tsk_a の優先度
strcpy( (char *)t_ctsk.dsname, "tsk_a" ); // tsk_a の名称
if( (objid = tk_cre_tsk( &t_ctsk )) <= E_OK ) { // tsk_a の生成
    tm_putstring(" *** Failed in the creation of tsk_a.\n");
    goto ERROR;
}
ObjID[TSK_A] = objid;
tm_putstring("*** tsk_a created.\n");

t_ctsk.task = tsk_b; // tsk_b の起動アドレス
t_ctsk.itskpri = 3; // tsk_b の優先度
strcpy( (char *)t_ctsk.dsname, "tsk_b" ); // tsk_b の名称
if( (objid = tk_cre_tsk( &t_ctsk )) <= E_OK ) { // tsk_b の生成
    tm_putstring(" *** Failed in the creation of tsk_b.\n");
    goto ERROR;
}
ObjID[TSK_B] = objid;
tm_putstring("*** tsk_b created.\n");

t_ctsk.task = rand_tsk; // rand_tsk の起動アドレス
t_ctsk.itskpri = 1; // rand_tsk の優先度
strncpy( (char *)t_ctsk.dsname, "rand_tsk", 8 ); // rand_tsk の名称
if( (objid = tk_cre_tsk( &t_ctsk )) <= E_OK ) { // rand_tsk の生成
    tm_putstring(" *** Failed in the creation of rand_tsk.\n");
    goto ERROR;
}
ObjID[RAND_TSK] = objid;
tm_putstring("*** rand_tsk created.\n");
if( tk_sta_tsk( ObjID[RAND_TSK], 0 ) != E_OK ) // rand_tsk の起動
    tm_putstring(" *** Failed in start of rand_tsk.\n");

PORTE.DDR.BYTE = 0xFF; // セグメントラインをアクティブに設定
PORT4.DDR.BYTE = 0xC0; // 左2桁のLEDをアクティブに設定

while( 1 ) wait( );
ERROR:
    return 0;
}

```

```

EXPORT void tsk_a(INT stacd, VP exinf)
{
    int i, j;
    // tk_wai_sem( ObjID[SEMID], 1, TMO_FEVR ); // セマフォの獲得
    PORT4.DR.BYTE = 0x00; // 7セグメントLEDを消灯
    PORTE.DR.BYTE = 0x63; // 上段のセグメントラインを点灯
    for( i=0 ; i<10 ; i++ ) {
        PORT4.DR.BYTE = 0x80; // 1桁目のLEDを点灯
        for( j=0 ; j<24000 ; j++ ) ; // 1msの待ち時間
    }
    // tk_sig_sem( ObjID[SEMID], 1 ); // セマフォの返却
    tk_ext_tsk( );
}

EXPORT void tsk_b(INT stacd, VP exinf)
{
    int i, j;
    // tk_wai_sem( ObjID[SEMID], 1, TMO_FEVR ); // セマフォの獲得
    PORT4.DR.BYTE = 0x00; // 7セグメントLEDを消灯
    PORTE.DR.BYTE = 0x5C; // 下段のセグメントラインを点灯
    for( i=0 ; i<10 ; i++ ) {
        PORT4.DR.BYTE = 0x40; // 2桁目のLEDを点灯
        for( j=0 ; j<24000 ; j++ ) ; // 1msの待ち時間
    }
    // tk_sig_sem( ObjID[SEMID], 1 ); // セマフォの返却
    tk_ext_tsk( );
}

EXPORT void rand_tsk(INT stacd, VP exinf)
{
    int i;
    while( 1 ) {
        tk_slp_tsk( i = (rand( ) & 7) + 6 ); // ランダム待ち
        tk_sta_tsk( ObjID[TSK_A], 0 ); // タスクの起動
        tk_slp_tsk( 21 - i ); // ランダム待ち
        tk_sta_tsk( ObjID[TSK_B], 0 ); // タスクの起動
    }
}

```

演習4 イベントフラグの演習

- ・ tsk_a、tsk_b の2つのタスクから構成
- ・ tsk_a から7セグメントLEDへの表示データをイベントフラグに設定
- ・ tsk_b はイベントフラグから7セグメントLEDの表示データを受け取り、表示
- ・ タスクの優先度が $tsk_a < tsk_b$ でなければデータの受け渡しが正しくできないことを確認する
- ・ イベントフラグのクリア方法を変更してもデータの受け渡しが正しくできることを確認する

04_eventflag.c

```
#include <tk/tkernel.h>
#include <tm/tmonitor.h>
#include <libstr.h>
#include <machine.h>
#include "iodefine.h"

EXPORT void tsk_a(INT stacd, VP exinf);
EXPORT void tsk_b(INT stacd, VP exinf);
EXPORT void irq15_hdr(UINT dintno);

typedef enum { FLGID, TSK_A, TSK_B, OBJ_KIND_NUM } OBJ_KIND;
EXPORT ID ObjID[OBJ_KIND_NUM];          // ID テーブル

EXPORT INT usermain( void )
{
    T_CFLG t_cflg;
    T_CTSK t_ctsk;
    T_DINT t_dint;
    ID objid;

    t_cflg.flgatr = TA_TFIFO | TA_WSGL | TA_DSNAME;    // イベントフラグ属性の設定
    t_cflg.iflgptn = 0;                                // イベントフラグの初期値の設定
    strcpy( (char *)t_cflg.dsname, "led_flg" );        // イベントフラグの名称
    if( (objid = tk_cre_flg( &t_cflg )) <= E_OK ) {    // イベントフラグの生成
        tm_putstring( "*** Failed in the creation of eventflag.¥n" );
        goto ERROR;
    }
    ObjID[FLGID] = objid;
    tm_putstring( "*** eventflag created.¥n" );

    t_ctsk.tskatr = TA_HLNG | TA_DSNAME;
    t_ctsk.stksz = 1024;
    t_ctsk.task = tsk_a;                                // tsk_a の起動アドレス
    t_ctsk.itskpri = 2;                                // tsk_a の優先度
    strcpy( (char *)t_ctsk.dsname, "tsk_a" );          // tsk_a の名称
    if( (objid = tk_cre_tsk( &t_ctsk )) <= E_OK ) {    // tsk_a の生成
        tm_putstring( "*** Failed in the creation of tsk_a.¥n" );
        goto ERROR;
    }
    ObjID[TSK_A] = objid;
    tm_putstring( "*** tsk_a created.¥n" );
}
```

```

t_ctsk.task = tsk_b;                                // tsk_b の起動アドレス
t_ctsk.itskpri = 1;                                  // tsk_b の優先度
strcpy( (char *)t_ctsk.dsname, "tsk_b" );           // tsk_b の名称
if( (objid = tk_cre_tsk( &t_ctsk )) <= E_OK ) {      // tsk_b の生成
    tm_putstring(" *** Failed in the creation of tsk_b.¥n");
    goto ERROR;
}
ObjID[TSK_B] = objid;
tm_putstring("*** tsk_b created.¥n");
if( tk_sta_tsk( ObjID[TSK_B], 0 ) != E_OK )          // tsk_b の起動
    tm_putstring(" *** Failed in start of tsk_b.¥n");

t_dint.intatr = TA_HLNG;
t_dint.inthdr = irq15_hdr;
if( tk_def_int( VECT( ICU, IRQ15 ), &t_dint ) != E_OK ) {
    tm_putstring(" *** Failed in the definition of irq15_hdr.¥n");
    goto ERROR;
}
tm_putstring("*** irq15_hdr defined.¥n");

PORTE.DDR.BYTE = 0xFF;                               // セグメントラインをアクティブに設定
PORT4.DDR.BYTE = 0xF0;                               // 7セグメントLEDをアクティブに設定

PORT0.ICR.BIT.B7 = 1;                                // P07(IRQ15)の入力バッファをON
ICU.IRQCR[15].BIT.IRQMD = 1;                         // IRQ15を立ち下がりエッジに設定
IR( ICU, IRQ15 ) = 0;                                // IRQ15の割り込み要求フラグをクリア
IPR( ICU, IRQ15 ) = 1;                                // IRQ15の割り込みレベルを1に設定
IEN( ICU, IRQ15 ) = 1;                                // IRQ15の割り込みを許可

while( 1 ) wait( );
ERROR:
return 0;
}

EXPORT void tsk_a(INT stacd, VP exinf)
{
    tm_putstring("*** tsk_a called tk_set_flg.¥n");
    tk_set_flg( ObjID[FLGID], (0x06<<8) + 0x80 );    // Call tk_set_flg -> 1***
    tm_putstring("*** tsk_a called tk_set_flg.¥n");
    tk_set_flg( ObjID[FLGID], (0x5B<<8) + 0x40 );    // Call tk_set_flg -> *2**
    tm_putstring("*** tsk_a called tk_set_flg.¥n");
    tk_set_flg( ObjID[FLGID], (0x4F<<8) + 0x20 );    // Call tk_set_flg -> **3*
    tm_putstring("*** tsk_a called tk_set_flg.¥n");
    tk_set_flg( ObjID[FLGID], (0x66<<8) + 0x10 );    // Call tk_set_flg -> ***4
    tk_ext_tsk( );                                    // tsk_aの終了
}

```

```
EXPORT void tsk_b(INT stacd, VP exinf)
{
    UINT flgptn;
    int i;
    tm_putstring("*** tsk_b started.¥n");
    while( 1 ) {
        tm_putstring("*** tsk_b is Waiting at eventflag.¥n");
        tk_wai_flg( ObjID[FLGID], 0xFFFF, TWF_ORW | TWF_CLR, &flgptn, TMO_FEVR );
        tm_putstring("*** tsk_b is Running.¥n");
        PORTE.DR.BYTE = flgptn >> 8;           // セグメントラインの設定
        PORT4.DR.BYTE = flgptn;                 // 7セグメント LED の点灯
        for( i=0 ; i<24000000 ; i++ ) ;        // 1000ms の待ち時間
        PORT4.DR.BYTE = 0x00;                   // 7セグメント LED の消灯
    }
}

EXPORT void irq15_hdr(UINT dintno)
{
    tk_sta_tsk( ObjID[TSK_A], 0 );             // tsk_a の起動
}
```


イベントフラグのスケジューリング

優先度：タスクA<タスクB

事象	カーネル	タスク A	タスク B
	0000 : イベントフラグ	RUNNING	WAITING
1 回目のtk_set_flg⇒	0000	READY	リターンパラメータ RUNNING 0680
tk_wai_flg⇒	0000	RUNNING	WAITING
2 回目のtk_set_flg⇒	0000	READY	RUNNING 5B40
tk_wai_flg⇒	0000	RUNNING	WAITING
3 回目のtk_set_flg⇒	0000	READY	RUNNING 4F20
tk_wai_flg⇒	0000	RUNNING	WAITING
4 回目のtk_set_flg⇒	0000	READY	RUNNING 6610
tk_wai_flg⇒	0000	RUNNING	WAITING
tk_ext_tsk⇒		DORMANT	

優先度：タスクA≧タスクB

事象	カーネル	タスク A	タスク B
	0000 : イベントフラグ	RUNNING	WAITING
1 回目のtk_set_flg⇒	0000		リターンパラメータ READY 0680
2 回目のtk_set_flg⇒	5B40		
3 回目のtk_set_flg⇒	5F60		
4 回目のtk_set_flg⇒	7F70		
tk_ext_tsk⇒		DORMANT	RUNNING
tk_wai_flg⇒	0000		7F70
tk_wai_flg⇒	0000		WAITING

演習5 メールボックスの演習

- ・ tsk_a、tsk_b の2つのタスクから構成
- ・ tsk_a から8個のLEDへの表示データをメッセージで送信
- ・ tsk_b はメールボックスから8個のLEDの表示データを受け取り、表示
- ・ タスクの優先度が $tsk_a < tsk_b$ でなければデータの受け渡しが正しくできないことを確認する
- ・ メッセージ本体の同期処理を追加すれば、データの受け渡しが正しくできることを確認する

05_mailbox.c

```
#include <tk/tkernel.h>
#include <tm/tmonitor.h>
#include <libstr.h>
#include <machine.h>
#include "idefine.h"

EXPORT void tsk_a(INT stacd, VP exinf);
EXPORT void tsk_b(INT stacd, VP exinf);
EXPORT void irq15_hdr(UINT dintno);

typedef struct user_msg {
    T_MSG msghead;           // メッセージキュー (OS 管理エリア)
    char *data;              // 送信データの先頭アドレス
} USER_MSG;

typedef enum { MBXID, TSK_A, TSK_B, OBJ_KIND_NUM } OBJ_KIND;
EXPORT ID ObjID[OBJ_KIND_NUM]; // ID テーブル

EXPORT INT usermain( void )
{
    T_CMBX t_cmbx;
    T_CTSK t_ctsk;
    T_DINT t_dint;
    ID objid;

    t_cmbx.mbxatr = TA_TFIFO | TA_MFIFO | TA_DSNAME; // メールボックス属性の設定
    strcpy( (char *)t_cmbx.dsname, "led_mbx" );      // メールボックスの名称
    if( (objid = tk_cre_mbx( &t_cmbx )) <= E_OK ) {   // メールボックスの生成
        tm_putstring( " *** Failed in the creation of mailbox. %n" );
        goto ERROR;
    }
    ObjID[MBXID] = objid;
    tm_putstring( " *** mailbox created. %n" );

    t_ctsk.tskatr = TA_HLNG | TA_DSNAME;
    t_ctsk.stksz = 1024;
    t_ctsk.task = tsk_a; // tsk_a の起動アドレス
    t_ctsk.itskpri = 2;   // tsk_a の優先度
    strcpy( (char *)t_ctsk.dsname, "tsk_a" );      // tsk_a の名称
    if( (objid = tk_cre_tsk( &t_ctsk )) <= E_OK ) { // tsk_a の生成
        tm_putstring( " *** Failed in the creation of tsk_a. %n" );
        goto ERROR;
    }
}
```

```

ObjID[TSK_A] = objid;
tm_putstring("*** tsk_a created.%n");
t_ctsk.task = tsk_b; // tsk_b の起動アドレス
strcpy( (char *)t_ctsk.dsname, "tsk_b" ); // tsk_b の名称
t_ctsk.itskpri = 1; // tsk_b の優先度
if( (objid = tk_cre_tsk( &t_ctsk )) <= E_OK ) { // tsk_b の生成
    tm_putstring(" *** Failed in the creation of tsk_b.%n");
    goto ERROR;
}
ObjID[TSK_B] = objid;
tm_putstring("*** tsk_b created.%n");
if( tk_sta_tsk( ObjID[TSK_B], 0 ) != E_OK ) // tsk_b の起動
    tm_putstring(" *** Failed in start of tsk_b.%n");

t_dint.intatr = TA_HLNG;
t_dint.inthdr = irq15_hdr;
if( tk_def_int( VECT( ICU, IRQ15 ), &t_dint ) != E_OK ) {
    tm_putstring(" *** Failed in the definition of irq15_hdr.%n");
    goto ERROR;
}
tm_putstring("*** irq15_hdr defined.%n");

PORTA.DR.BYTE = 0xFF; // LED を消灯に設定
PORTA.DDR.BYTE = 0xFF; // LED をアクティブに設定
PORT0.ICR.BIT.B7 = 1; // P07(IRQ15) の入力バッファを ON
ICU.IRQCR[15].BIT.IRQMD = 1; // IRQ15 を立ち下がりエッジに設定
IR( ICU, IRQ15 ) = 0; // IRQ15 の割り込み要求フラグをクリア
IPR( ICU, IRQ15 ) = 1; // IRQ15 の割り込みレベルを 1 に設定
IEN( ICU, IRQ15 ) = 1; // IRQ15 の割り込みを許可

while( 1 ) wait( );
ERROR:
    return 0;
}

EXPORT void tsk_a(INT stacd, VP exinf)
{
    LOCAL USER_MSG msg;
    msg.data = "abc"; // 文字列 "abc" を設定
    tm_putstring("*** tsk_a called tk_snd_mbx.%n");
    tk_snd_mbx( ObjID[MBXID], (T_MSG *)&msg ); // Call tk_snd_mbx
    // tk_slp_tsk( TMO_FEVR ); // 送信メッセージの処理待ち
    msg.data = "ABC"; // 文字列 "ABC" を設定
    tm_putstring("*** tsk_a called tk_snd_mbx.%n");
    tk_snd_mbx( ObjID[MBXID], (T_MSG *)&msg ); // Call tk_snd_mbx
    // tk_slp_tsk( TMO_FEVR ); // 送信メッセージの処理待ち
    msg.data = "01234"; // 文字列 "01234" を設定
    tm_putstring("*** tsk_a called tk_snd_mbx.%n");
    tk_snd_mbx( ObjID[MBXID], (T_MSG *)&msg ); // Call tk_snd_mbx
    // tk_slp_tsk( TMO_FEVR ); // 送信メッセージの処理待ち
    tk_ext_tsk( ); // tsk_a の終了
}

```

```

EXPORT void tsk_b(INT stacd, VP exinf)
{
  USER_MSG *msg;
  int i, j;
  tm_putstring("*** tsk_b started.¥n");
  while( 1 ) {
    tm_putstring("*** tsk_b is Waiting at mailbox.¥n");
    tk_rcv_mbx( ObjID[MBXID], (T_MSG **)&msg, TMO_FEVR );
    tm_putstring("*** tsk_b is Running.¥n");
    for( i=0 ; msg->data[i]!='¥0' ; i++ ) { // 文字列長のループ
      PORTA.DR.BYTE = ~msg->data[i]; // LED に文字コードを表示
      for( j=0 ; j<24000000 ; j++ ) ; // 1000ms の待ち時間
    }
    PORTA.DR.BYTE = 0xFF; // LED を消灯に設定
    // tk_wup_tsk( ObjID[TSK_A] ); // メッセージの処理完了通知
  }
}

EXPORT void irq15_hdr(UINT dintno)
{
  tk_sta_tsk( ObjID[TSK_A], 0 ); // tsk_a の起動
}

```

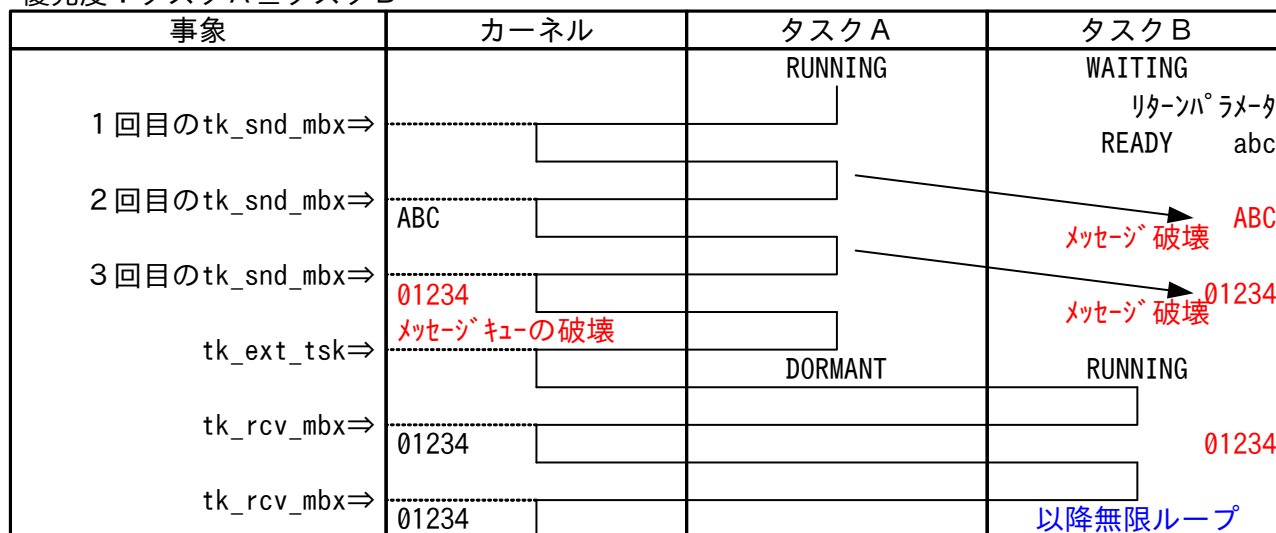
メールボックスのスケジューリング

優先度：タスクA<タスクB

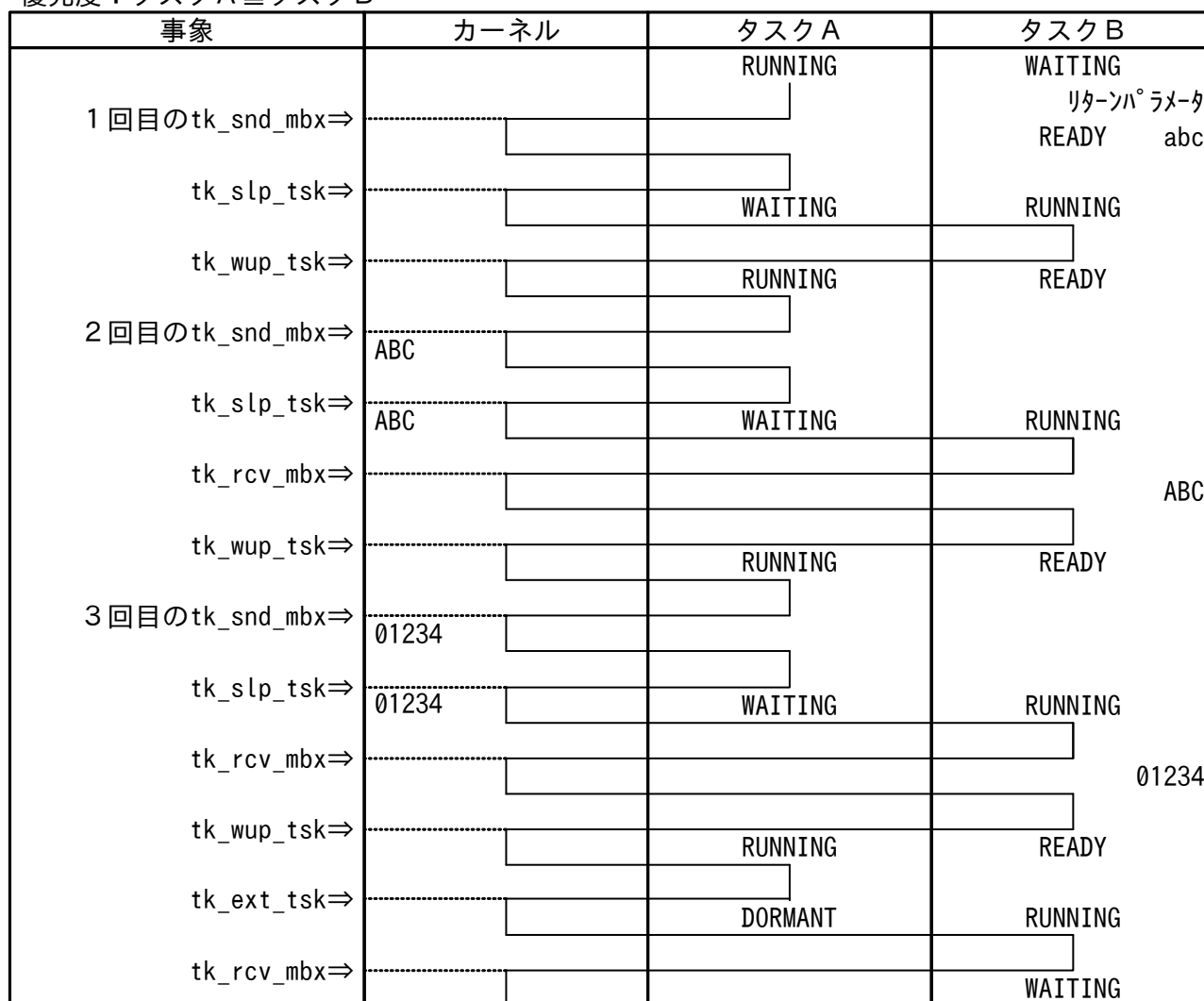
事象	カーネル	タスク A	タスク B
		RUNNING	WAITING
1 回目のtk_snd_mbx⇒		READY	リターンパラメータ RUNNING abc
tk_rcv_mbx⇒		RUNNING	WAITING
2 回目のtk_snd_mbx⇒		READY	RUNNING ABC
tk_rcv_mbx⇒		RUNNING	WAITING
3 回目のtk_snd_mbx⇒		READY	RUNNING 01234
tk_rcv_mbx⇒		RUNNING	WAITING
tk_ext_tsk⇒		DORMANT	

メールボックスのスケジューリング

優先度：タスクA ≧ タスクB



優先度：タスクA ≧ タスクB



演習6 ミューテックスの演習

- ・ tsk_a、tsk_b の2つのタスクと両者を周期的に起動する rand_tsk の3つから構成
- ・ tsk_a は最左桁の7セグメントLEDの上段を点灯する
- ・ tsk_b は2桁目の7セグメントLEDの下段を点灯する
- ・ 排他制御を行わないと7セグメントLEDの表示が正しく行えないことを確認する
- ・ ミューテックスにより排他制御を行えば7セグメントLEDの表示を正しく行えることを確認する

06_mutex.c

```
#include <tk/tkernel.h>
#include <tm/tmonitor.h>
#include <libstr.h>
#include <machine.h>
#include "idefine.h"

EXPORT void tsk_a(INT stacd, VP exinf);
EXPORT void tsk_b(INT stacd, VP exinf);
EXPORT void rand_tsk(INT stacd, VP exinf);
EXPORT unsigned int rand(void);

typedef enum { MTXID, TSK_A, TSK_B, RAND_TSK, OBJ_KIND_NUM } OBJ_KIND;
EXPORT ID ObjID[OBJ_KIND_NUM];          // ID テーブル

EXPORT INT usermain( void )
{
    T_CMTX t_cmtx;
    T_CTSK t_ctsk;
    ID objid;

    t_cmtx.mtxatr = TA_INHERIT | TA_DSNAME;          // 優先度継承プロトコル
    // t_cmtx.mtxatr = TA_CEILING;                  // 優先度上限プロトコル
    // t_cmtx.ceilpri = 2;                          // 上限優先度の指定
    strcpy( (char *)t_cmtx.dsname, "led_mtx" );      // ミューテックスの名称
    if( (objid = tk_cre_mtx( &t_cmtx )) <= E_OK ) {   // ミューテックスの生成
        tm_putstring(" *** Failed in the creation of mutex.¥n");
        goto ERROR;
    }
    ObjID[MTXID] = objid;
    tm_putstring(" *** mutex created.¥n");
}
```

```

t_ctsk.tskatr = TA_HLNG | TA_RNG0;
t_ctsk.stksz = 1024;
t_ctsk.task = tsk_a; // tsk_a の起動アドレス
t_ctsk.itskpri = 2; // tsk_a の優先度
strcpy( (char *)t_ctsk.dsname, "tsk_a" ); // tsk_a の名称
if( (objid = tk_cre_tsk( &t_ctsk )) <= E_OK ) { // tsk_a の生成
    tm_putstring(" *** Failed in the creation of tsk_a.\n");
    goto ERROR;
}
ObjID[TSK_A] = objid;
tm_putstring("*** tsk_a created.\n");

t_ctsk.task = tsk_b; // tsk_b の起動アドレス
t_ctsk.itskpri = 3; // tsk_b の優先度
strcpy( (char *)t_ctsk.dsname, "tsk_b" ); // tsk_b の名称
if( (objid = tk_cre_tsk( &t_ctsk )) <= E_OK ) { // tsk_b の生成
    tm_putstring(" *** Failed in the creation of tsk_b.\n");
    goto ERROR;
}
ObjID[TSK_B] = objid;
tm_putstring("*** tsk_b created.\n");

t_ctsk.task = rand_tsk; // rand_tsk の起動アドレス
t_ctsk.itskpri = 1; // rand_tsk の優先度
strncpy( (char *)t_ctsk.dsname, "rand_tsk", 8 ); // rand_tsk の名称
if( (objid = tk_cre_tsk( &t_ctsk )) <= E_OK ) { // rand_tsk の生成
    tm_putstring(" *** Failed in the creation of rand_tsk.\n");
    goto ERROR;
}
ObjID[RAND_TSK] = objid;
tm_putstring("*** rand_tsk created.\n");
if( tk_sta_tsk( ObjID[RAND_TSK], 0 ) != E_OK ) // rand_tsk の起動
    tm_putstring(" *** Failed in start of rand_tsk.\n");

PORTE.DDR.BYTE = 0xFF; // セグメントラインをアクティブに設定
PORT4.DDR.BYTE = 0xC0; // 左2桁のLEDをアクティブに設定

while( 1 ) wait( );
ERROR:
    return 0;
}

```

```

EXPORT void tsk_a(INT stacd, VP exinf)
{
    int i, j;
    // tk_loc_mtx( ObjID[MTXID], TMO_FEVR ); // ミューテックスの獲得
    PORT4.DR.BYTE = 0x00; // 7セグメントLEDを消灯
    PORTE.DR.BYTE = 0x63; // 上段のセグメントラインを点灯
    for( i=0 ; i<10 ; i++ ) {
        PORT4.DR.BYTE = 0x80; // 1桁目のLEDを点灯
        for( j=0 ; j<24000 ; j++ ) ; // 1msの待ち時間
    }
    // tk_unl_mtx( ObjID[MTXID] ); // ミューテックスの返却
    tk_ext_tsk( );
}

EXPORT void tsk_b(INT stacd, VP exinf)
{
    int i, j;
    // tk_loc_mtx( ObjID[MTXID], TMO_FEVR ); // ミューテックスの獲得
    PORT4.DR.BYTE = 0x00; // 7セグメントLEDを消灯
    PORTE.DR.BYTE = 0x5C; // 下段のセグメントラインを点灯
    for( i=0 ; i<10 ; i++ ) {
        PORT4.DR.BYTE = 0x40; // 2桁目のLEDを点灯
        for( j=0 ; j<24000 ; j++ ) ; // 1msの待ち時間
    }
    // tk_unl_mtx( ObjID[MTXID] ); // ミューテックスの返却
    tk_ext_tsk( );
}

EXPORT void rand_tsk(INT stacd, VP exinf)
{
    int i;
    while( 1 ) {
        tk_slp_tsk( i = (rand( ) & 7) + 6 ); // ランダム待ち
        tk_sta_tsk( ObjID[TSK_A], 0 ); // タスクの起動
        tk_slp_tsk( 21 - i ); // ランダム待ち
        tk_sta_tsk( ObjID[TSK_B], 0 ); // タスクの起動
    }
}

```


演習7 メッセージバッファの演習

- ・ tsk_a、tsk_b の2つのタスクから構成
- ・ tsk_a から7セグメントLEDへの表示データをメッセージバッファに設定
- ・ tsk_b はメッセージバッファから7セグメントLEDの表示データを受け取り、表示
- ・ タスクの優先度に関係なくデータの受け渡しが正しくできることを確認する
- ・ メッセージバッファのサイズに関係なく、データの受け渡しが正しくできることを確認する

07_messagebuffer.c

```
#include <tk/tkernel.h>
#include <tm/tmonitor.h>
#include <libstr.h>
#include <machine.h>
#include "idefine.h"

EXPORT void tsk_a(INT stacd, VP exinf);
EXPORT void tsk_b(INT stacd, VP exinf);
EXPORT void irq15_hdr(UINT dintno);

typedef enum { MBFID, TSK_A, TSK_B, OBJ_KIND_NUM } OBJ_KIND;
EXPORT ID ObjID[OBJ_KIND_NUM];          // ID テーブル

EXPORT INT usermain( void )
{
    T_CMBF t_cmbf;
    T_CTSK t_ctsk;
    T_DINT t_dint;
    ID objid;

    t_cmbf.mbfatr = TA_TFIFO | TA_DSNAME;    // メッセージバッファ属性の設定
    t_cmbf.bufsz = (4 + 4) * 1;              // メッセージバッファのサイズの設定
    // ( 管理エリア+データ長 ) × 個数
    t_cmbf.maxmsz = 2;                      // メッセージの最大長の設定
    strcpy( (char *)t_cmbf.dsname, "led_mbf" );    // メッセージバッファの名称
    if( (objid = tk_cre_mbf( &t_cmbf )) <= E_OK ) { // メッセージバッファの生成
        tm_putstring( " *** Failed in the creation of messagebuffer.¥n" );
        goto ERROR;
    }
    ObjID[MBFID] = objid;
    tm_putstring( " *** messagebuffer created.¥n" );

    t_ctsk.tskatr = TA_HLNG | TA_RNG0;
    t_ctsk.stksz = 1024;
    t_ctsk.task = tsk_a;                    // tsk_a の起動アドレス
    t_ctsk.itskpri = 1;                     // tsk_a の優先度
    strcpy( (char *)t_ctsk.dsname, "tsk_a" );    // tsk_a の名称
    if( (objid = tk_cre_tsk( &t_ctsk )) <= E_OK ) { // tsk_a の生成
        tm_putstring( " *** Failed in the creation of tsk_a.¥n" );
        goto ERROR;
    }
    ObjID[TSK_A] = objid;
    tm_putstring( " *** tsk_a created.¥n" );
}
```

```

t_ctsk.task = tsk_b; // tsk_b の起動アドレス
t_ctsk.itskpri = 2; // tsk_b の優先度
strcpy( (char *)t_ctsk.dsname, "tsk_b" ); // tsk_b の名称
if( (objid = tk_cre_tsk( &t_ctsk )) <= E_OK ) { // tsk_b の生成
    tm_putstring( " *** Failed in the creation of tsk_b.¥n" );
    goto ERROR;
}
ObjID[TSK_B] = objid;
tm_putstring( " *** tsk_b created.¥n" );
if( tk_sta_tsk( ObjID[TSK_B], 0 ) != E_OK ) // tsk_b の起動
    tm_putstring( " *** Failed in start of tsk_b.¥n" );

t_dint.intatr = TA_HLNG;
t_dint.inthdr = irq15_hdr;
if( tk_def_int( VECT( ICU, IRQ15 ), &t_dint ) != E_OK ) {
    tm_putstring( " *** Failed in the definition of irq15_hdr.¥n" );
    goto ERROR;
}
tm_putstring( " *** irq15_hdr defined.¥n" );

PORTE.DDR.BYTE = 0xFF; // セグメントラインをアクティブに設定
PORT4.DDR.BYTE = 0xF0; // 7セグメントLEDをアクティブに設定

PORT0.ICR.BIT.B7 = 1; // P07(IRQ15)の入力バッファをON
ICU.IRQCR[15].BIT.IRQMD = 1; // IRQ15を立ち下がりエッジに設定
IR( ICU, IRQ15 ) = 0; // IRQ15の割り込み要求フラグをクリア
IPR( ICU, IRQ15 ) = 1; // IRQ15の割り込みレベルを1に設定
IEN( ICU, IRQ15 ) = 1; // IRQ15の割り込みを許可

while( 1 ) wait( );
ERROR:
return 0;
}

EXPORT void tsk_a(INT stacd, VP exinf)
{
    UB data[2];
    tm_putstring( " *** tsk_a called tk_snd_mbf.¥n" );
    data[0] = 0x06; data[1] = 0x80;
    tk_snd_mbf( ObjID[MBFID], data, 2, TMO_FEVR ); // Call tk_snd_mbf -> 1***
    tm_putstring( " *** tsk_a called tk_snd_mbf.¥n" );
    data[0] = 0x5B; data[1] = 0x40;
    tk_snd_mbf( ObjID[MBFID], data, 2, TMO_FEVR ); // Call tk_snd_mbf -> *2**
    tm_putstring( " *** tsk_a called tk_snd_mbf.¥n" );
    data[0] = 0x4F; data[1] = 0x20;
    tk_snd_mbf( ObjID[MBFID], data, 2, TMO_FEVR ); // Call tk_snd_mbf -> **3*
    tm_putstring( " *** tsk_a called tk_snd_mbf.¥n" );
    data[0] = 0x66; data[1] = 0x10;
    tk_snd_mbf( ObjID[MBFID], data, 2, TMO_FEVR ); // Call tk_snd_mbf -> ***4
    tk_ext_tsk( ); // tsk_aの終了
}

```

```

EXPORT void tsk_b(INT stacd, VP exinf)
{
  UB data[2];
  int i;
  tm_putstring("*** tsk_b started.¥n");
  while( 1 ) {
    tm_putstring("*** tsk_b is Waiting at messagebuffer.¥n");
    tk_rcv_mbf( ObjID[MBFID], data, TMO_FEVR );      // tk_rcv_mbf
    tm_putstring("*** tsk_b is Running.¥n");
    PORTE.DR.BYTE = data[0];                        // セグメントラインの設定
    PORT4.DR.BYTE = data[1];                        // 7セグメントLEDの点灯
    for( i=0 ; i<24000000 ; i++ ) ;                // 1000msの待ち時間
    PORT4.DR.BYTE = 0x00;                          // 7セグメントLEDの消灯
  }
}

EXPORT void irq15_hdr(UINT dintno)
{
  tk_sta_tsk( ObjID[TSK_A], 0 );                  // tsk_aの起動
}

```

メッセージバッファのスケジューリング

優先度：タスクA ≧ タスクB

事象	カーネル	タスク A	タスク B
	---- : メッセージバッファ	RUNNING	WAITING
1回目のtk_snd_mbf⇒	-----		リターンパラメータ READY 0680
2回目のtk_snd_mbf⇒	5B40		
3回目のtk_snd_mbf⇒	5B40	WAITING 4F20	RUNNING
tk_rcv_mbf⇒	4F20	RUNNING	READY 5B40
4回目のtk_snd_mbf⇒	4F20	WAITING 6610	RUNNING
tk_rcv_mbf⇒	6610	RUNNING	READY 4F20
tk_ext_tsk⇒		DORMANT	RUNNING
tk_rcv_mbf⇒	-----		6610
tk_rcv_mbf⇒	-----		WAITING

メッセージバッファのスケジューリング

優先度：タスクA<タスクB

事象	カーネル	タスク A	タスク B
	---- : メッセージ バッファ	RUNNING	WAITING リターンパ ラメータ
1 回目のtk_snd_mbf⇒	-----	READY	RUNNING 0680
tk_rcv_mbf⇒	-----	RUNNING	WAITING
2 回目のtk_snd_mbf⇒	-----	READY	RUNNING 5B40
tk_rcv_mbf⇒	-----	RUNNING	WAITING
3 回目のtk_snd_mbf⇒	-----	READY	RUNNING 4F20
tk_rcv_mbf⇒	-----	RUNNING	WAITING
4 回目のtk_snd_mbf⇒	-----	READY	RUNNING 6610
tk_rcv_mbf⇒	-----	RUNNING	WAITING
tk_ext_tsk⇒	-----	DORMANT	

優先度：タスクA>タスクB

事象	カーネル	タスク A	タスク B
		RUNNING	WAITING リターンパ ラメータ
1 回目のtk_snd_mbf⇒	-----	READY 0680	
2 回目のtk_snd_mbf⇒	-----	WAITING 5B40	RUNNING
tk_rcv_mbf⇒	-----	RUNNING	READY 5B40
3 回目のtk_snd_mbf⇒	-----	WAITING 4F20	RUNNING
tk_rcv_mbf⇒	-----	RUNNING	READY 4F20
4 回目のtk_snd_mbf⇒	-----	WAITING 6610	RUNNING
tk_rcv_mbf⇒	-----	RUNNING	READY 6610
tk_ext_tsk⇒	-----	DORMANT	RUNNING
tk_rcv_mbf⇒	-----		WAITING

演習8 固定長メモリのプールの演習

- ・ tsk_a、tsk_b の2つのタスクから構成
- ・ tsk_a は固定長メモリプールからメッセージ用のメモリブロックを獲得
- ・ tsk_a から8個のLEDへの表示データをメッセージで送信
- ・ tsk_b はメールボックスから8個のLEDの表示データを受け取り、表示
- ・ tsk_b はメッセージ用のメモリブロックを返却
- ・ タスクの優先度に関係なくデータの受け渡しが正しくできることを確認する
- ・ メモリブロック数に関係なく、データの受け渡しが正しくできることを確認する

08_memorypool.c

```
#include <tk/tkernel.h>
#include <tm/tmonitor.h>

#include <machine.h>
#include "iodefine.h"

EXPORT void tsk_a(INT stacd, VP exinf);
EXPORT void tsk_b(INT stacd, VP exinf);
EXPORT void irq15_hdr(UINT dintno);

typedef struct user_msg {
    T_MSG msghead;           // メッセージキュー (OS 管理エリア)
    char *data;              // 送信データの先頭アドレス
} USER_MSG;

typedef enum { MPFID, MBXID, TSK_A, TSK_B, OBJ_KIND_NUM } OBJ_KIND;
EXPORT ID ObjID[OBJ_KIND_NUM]; // ID テーブル

EXPORT INT usermain( void )
{
    T_CMPF t_cmpf;
    T_CMBX t_cmbx;
    T_CTSK t_ctsk;
    T_DINT t_dint;
    ID objid;

    t_cmpf.mpfatr = TA_TFIFO | TA_DSNAME; // 固定長メモリプール属性の設定
    t_cmpf.mpfcnt = 2; // メモリブロックのブロック数の設定
    t_cmpf.blfsz = sizeof( USER_MSG ); // メモリブロックのサイズの設定
    strcpy( (char *)t_cmpf.dsname, "msg_mbf" ); // 固定長メモリアプールの名称
    if( (objid = tk_cre_mpf( &t_cmpf )) <= E_OK ) { // 固定長メモリアプールの生成
        tm_putstring( " *** Failed in the creation of memorypool fixed. %n" );
        goto ERROR;
    }
    ObjID[MPFID] = objid;
    tm_putstring( " *** memorypool fixed created. %n" );
}
```

```

t_cmbx.mbxatr = TA_TFIFO | TA_MFIFO | TA_DSNAME; // メールボックス属性の設定
if( (objid = tk_cre_mbx( &t_cmbx )) <= E_OK ) { // メールボックスの生成
strcpy( (char *)t_cmbx.dsname, "led_mbx" ); // メールボックスの名称
tm_putstring(" *** Failed in the creation of mailbox.¥n");
goto ERROR;
}
ObjID[MBXID] = objid;
tm_putstring("*** mailbox created.¥n");

t_ctsk.tskatr = TA_HLNG | TA_DSNAME;
t_ctsk.stksz = 1024;
t_ctsk.task = tsk_a; // tsk_a の起動アドレス
t_ctsk.itstkpri = 2; // tsk_a の優先度
strcpy( (char *)t_ctsk.dsname, "tsk_a" ); // tsk_a の名称
if( (objid = tk_cre_tsk( &t_ctsk )) <= E_OK ) { // tsk_a の生成
tm_putstring(" *** Failed in the creation of tsk_a.¥n");
goto ERROR;
}
ObjID[TSK_A] = objid;
tm_putstring("*** tsk_a created.¥n");

t_ctsk.task = tsk_b; // tsk_b の起動アドレス
t_ctsk.itstkpri = 1; // tsk_b の優先度
strcpy( (char *)t_ctsk.dsname, "tsk_b" ); // tsk_b の名称
if( (objid = tk_cre_tsk( &t_ctsk )) <= E_OK ) { // tsk_b の生成
tm_putstring(" *** Failed in the creation of tsk_b.¥n");
goto ERROR;
}
ObjID[TSK_B] = objid;
tm_putstring("*** tsk_b created.¥n");
if( tk_sta_tsk( ObjID[TSK_B], 0 ) != E_OK ) // tsk_b の起動
tm_putstring(" *** Failed in start of tsk_b.¥n");

t_dint.intatr = TA_HLNG;
t_dint.inthdr = irq15_hdr;
if( tk_def_int( VECT( ICU, IRQ15 ), &t_dint ) != E_OK ) {
tm_putstring(" *** Failed in the definition of irq15_hdr.¥n");
goto ERROR;
}
tm_putstring("*** irq15_hdr defined.¥n");

PORTA.DR.BYTE = 0xFF; // LED を消灯に設定
PORTA.DDR.BYTE = 0xFF; // LED をアクティブに設定
PORT0.ICR.BIT.B7 = 1; // P07(IRQ15) の入力バッファを ON
ICU.IRQCR[15].BIT.IRQMD = 1; // IRQ15 を立ち下がりエッジに設定
IR( ICU, IRQ15 ) = 0; // IRQ15 の割り込み要求フラグをクリア
IPR( ICU, IRQ15 ) = 1; // IRQ15 の割り込みレベルを 1 に設定
IEN( ICU, IRQ15 ) = 1; // IRQ15 の割り込みを許可
while( 1 ) wait( );

ERROR:
return 0;
}

```

```

EXPORT void tsk_a(INT stacd, VP exinf)
{
    USER_MSG *msg;
    tk_get_mpf( ObjID[MPFID], (VP *)&msg, TMO_FEVR ); // メッセージ本体を獲得
    tm_putstring("*** tsk_a got memory block.¥n");
    msg->data = "abc"; // 文字列 "abc" を設定
    tm_putstring("*** tsk_a called tk_snd_mbx.¥n");
    tk_snd_mbx( ObjID[MBXID], (T_MSG *)&msg ); // Call tk_snd_mbx
    tk_get_mpf( ObjID[MPFID], (VP *)&msg, TMO_FEVR ); // メッセージ本体を獲得
    tm_putstring("*** tsk_a got memory block.¥n");
    msg->data = "ABC"; // 文字列 "ABC" を設定
    tm_putstring("*** tsk_a called tk_snd_mbx.¥n");
    tk_snd_mbx( ObjID[MBXID], (T_MSG *)&msg ); // Call tk_snd_mbx
    tk_get_mpf( ObjID[MPFID], (VP *)&msg, TMO_FEVR ); // メッセージ本体を獲得
    tm_putstring("*** tsk_a got memory block.¥n");
    msg->data = "01234"; // 文字列 "01234" を設定
    tm_putstring("*** tsk_a called tk_snd_mbx.¥n");
    tk_snd_mbx( ObjID[MBXID], (T_MSG *)&msg ); // Call tk_snd_mbx
    tk_ext_tsk( ); // tsk_a の終了
}

EXPORT void tsk_b(INT stacd, VP exinf)
{
    USER_MSG *msg;
    int i, j;
    tm_putstring("*** tsk_b started.¥n");
    while( 1 ) {
        tm_putstring("*** tsk_b is Waiting at mailbox.¥n");
        tk_rcv_mbx( ObjID[MBXID], (T_MSG *)&msg, TMO_FEVR );
        tm_putstring("*** tsk_b is Running.¥n");
        for( i=0 ; msg->data[i]!='¥0' ; i++ ) { // 文字列長のループ
            PORTA.DR.BYTE = ~msg->data[i]; // LED に文字コードを表示
            for( j=0 ; j<24000000 ; j++ ) ; // 1000ms の待ち時間
        }
        PORTA.DR.BYTE = 0xFF; // LED を消灯に設定
        tm_putstring("*** tsk_b released memory block.¥n");
        tk_rel_mpf( ObjID[MPFID], msg ); // メッセージ本体を返却
    }
}

EXPORT void irq15_hdr(UINT dintno)
{
    tk_sta_tsk( ObjID[TSK_A], 0 ); // tsk_a の起動
}

```

トロンフォーラム
【実習】 μ T-**Kernel** 入門（協力：ルネサス エレクトロニクス）
プログラミング演習

2016 年 6 月 23 日発行

発行所
トロンフォーラム
(YRP ユビキタス・ネットワーキング研究所内)
〒141-0031 東京都品川区西五反田 2-12-3 第一誠実ビル 9F
URL: <http://www.tron.org/ja/>
TEL:03-5437-0572(代表) FAX:03-5437-2399(代表)

本テキストは、クリエイティブ・コモンズ 表示 - 継承 4.0 国際 ライセンスの下に提供されています。



<https://creativecommons.org/licenses/by-sa/4.0/deed.ja>

Copyright ©2016 TRON Forum

【ご注意およびお願い】

- 1.本テキストの中で第三者が著作権等の権利を有している箇所については、利用者の方が当該第三者から利用許諾を得てください。
- 2.本テキストの内容については、その正確性、網羅性、特定目的への適合性等、一切の保証をしないほか、本テキストを利用したことにより損害が生じても著者は責任を負いません。
- 3.本テキストをご利用いただく際、可能であれば office@tron.org までご利用者のお名前、ご所属、ご連絡先メールアドレスをご連絡いただければ幸いです。

トロンフォーラム©2016

Printed in Japan